A Visual, Open-Ended Approach to Prototyping Ubiquitous Computing Applications

Zoé Drey
Thales/INRIA
Talence, France
zoe.drey@inria.fr

Charles Consel
ENSEIRB/INRIA
Talence, France
charles.consel@inria.fr

***Abstract*—By nature, ubiquitous computing applications are intimately intertwined with users' everyday life. This situation is challenging because it requires to make the development of applications accessible to end-users. Furthermore, ubiquitous computing consists of a variety of areas, including home automation and assisted living, raising a need for an open-ended approach.**

We present Pantagruel, a visual programming language that is end-user oriented. Our approach is open-ended in that Pantagruel integrates a language to describe a ubiquitous computing environment. Such description takes the form of a taxonomy, defining the entities relevant to a given ubiquitous computing area. This description serves as a parameter to a sensor-controller-actuator development paradigm. The orchestration of area-specific entities is supported by high-level constructs, customized with respect to taxonomical information.

We have implemented a visual environment to prototyping ubiquitous computing applications. Furthermore, we have developed a compiler for Pantagruel that targets a domain-specific middleware. Our environment leverages a 2D renderer to enable the simulation and of applications. We successfully simulated a range of applications in various ubiquitous computing areas, such as home automation, assisted living and building management.

***Keywords*-End-User Visual Language, Ubiquitous Computing Prototyping, Programming paradigms for Ubiquitous Systems.**

I. Introduction

Ubiquitous computing aims to address end-user needs pertaining to various areas such as assisted living, home automation or energy management. Because it requires expertise in many fields (*e.g.,* networking, multimedia, and systems), programming ubiquitous computing systems is very challenging. Even more challenging is the fact that this programming must be made accessible to end-users because ubiquitous computing applications are intimately involved in our everyday life. Also, the spectrum of potential application areas requires the development process to be open-ended, enabling new entities, whether devices or components, to be integrated.

This paper presents Pantagruel, an expressive approach to developing orchestration logic of an open-ended set of entities. A Pantagruel program is parameterized with respect to a *taxonomy* of entities describing a ubiquitous computing environment. Our visual programming environment includes a simulator that allows programs to be tested prior to their deployment.

Our approach consists of a two-step process: (1) a ubiquitous computing environment is described in terms of its constituent entities, their functionalities and their properties; (2) this description takes the form of a taxonomy that drives the development of an application.

The environment description allows our approach to be instantiated with respect to a given application area. This description defines the classes of entities that are relevant to the target area. For each class, it specifies an interface to access its functionalities. Because the orchestration logic is written with respect to the environment description, entities are combined in compliance with their description. To facilitate the programming of the orchestration logic, we have developed a visual tool that uses a sensor-controller-actuator paradigm. An orchestration logic collects context data from sensors, combines them with a controller, and reacts by triggering actuators. We assessed the usability of this paradigm with a successfull user study conducted with 18 non-programmer participants. Furthermore, our visual programming environment offers the developer an interface that is customized with respect to the environment description. Information about the environment entities is exploited to guide the programmer in defining sensor-controller-actuator rules. Finally, Pantagruel programs are compiled and executed on a platform dedicated to ubiquitous computing applications.

II. Our approach

We now describe the key features of our approach.

A. An open-ended approach

We define a novel approach to visual programming of ubiquitous computing applications that is parameterized with respect to the description of a ubiquitous computing environment. In doing so, our approach addresses a range of areas.

An environment description consists of declarations of entity classes, each of which characterizes a collection of entities that share common functionalities. The declaration of an entity class lists how to interact with entities belonging to this class. The generality of these declarations makes it possible to abstract over a range of variations, enabling the re-use of an environment description.

Specifically, the declaration of an entity class consists of attributes defining a context and methods accessing the entity

functionalities. A context element may either be constant, external or applicative, according to the kind of information it exposes. Let us introduce each kind of context.

An information may be assigned once and for all to an entity instance at deployment time. This first kind of context information is said to be *constant*. A constant is used to designate the location of an entity instance, its name, or any other information that remains unchanged throughout the life-cycle of the ubiquitous computing environment. The environment continuously evolves over time and is periodically sensed by entities, whether hardware or software, as is done by a motion sensor, for example. This second kind of context information is said to be *external*. Finally, context information can also be assigned values during the application execution. This third kind of context information is called *applicative*.

These three kinds of contexts are respectively interfaced via *constant*, *volatile*, and *write* attributes. Furthermore, attributes are typed to make their purpose explicit. As for methods, they are the interface declarations to the actions provided by an entity, that can be viewed as an actuator. They can also affect the applicative context. For example, the zoom method of a webcam may update the device status.

The declarations form a taxonomy describing a given environment. This is illustrated by the taxonomy extract for the assisted-living area displayed in Figure 1.

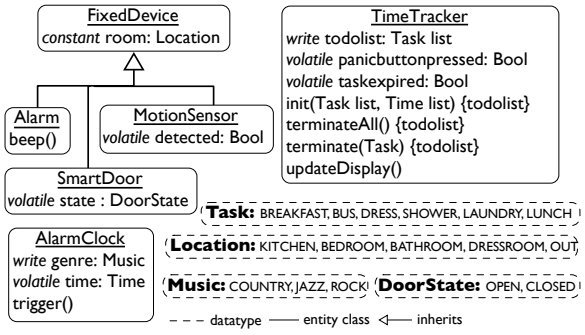

Figure 1. An extract of a taxonomy for assisted living

A given taxonomy is used to define concrete environments by instantiating entity classes, as will be illustrated in our demonstration, summarized in Section III-B.

B. A visual development environment

We now present our visual environment dedicated to the development of orchestration rules. To implement our paradigm, the Pantagruel development environment offers a panel divided in three columns: sensors, controllers, and actuators. This panel is shown in Figure 2. To develop an application, the programmer starts by identifying entities that need to be orchestrated in this application. This is achieved in the visual language by creating sections accross the 3-column panel, each representing an entity available in the concrete environment. Then, the developer defines conditions on context elements provided by the entity interface in the sensor column, combining them in the controller column, and triggering actions on an entity in the actuator column. For readability, rules are numbered in the controller column (*e.g.,*R1). A key feature of our approach is to drive the development of orchestration rules with respect to an environment description. In doing so, the development environment provides the programmer with contextual menus and on-the-fly verification to guide the definition of rules.

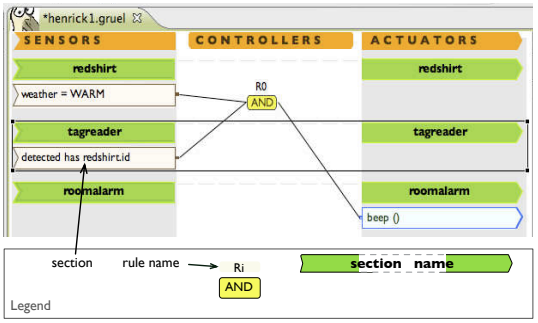

Figure 2. Editing of orchestration rules

C. A compiler towards a ubicomp platform

To allow Pantagruel programs to be simulated and tested, we have developed a compiler that leverages an architecture description language (ADL), dedicated to distributed systems, named DiaSpec [1]. Given an architecture description, the compiler for this ADL generates a dedicated programming framework in Java, which provides extensive support to discover and interact with distributed entities. The compilation process of Pantagruel consists of two stages: (1) an environment description is translated into a DiaSpec description (2) orchestration rules are compiled into Java code, supported by a DiaSpec-generated programming framework. To allow Pantagruel applications to be tested, we leverage an existing entity library that implements the entity action interfaces. The overall process is illustrated in Figure 3 and described in [2].

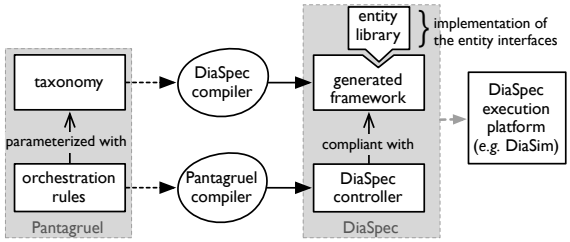

Figure 3. Compilation process of Pantagruel

The applications can be executed in a simulator, called DiaSim [3]. This simulator provides 2D rendering as illustrated by Figure 5. The simulated entities are displayed in a

2D model of the physical environment, and messages appear above the entities when sensing or actuating is performed.

III. Demonstration

The goal of this demonstration is to show how to program a ubiquitous computing application with Pantagruel. During this demonstration, we will instantiate a concrete environment for the assisted-living area, create Pantagruel orchestration rules, compile them, and simulate the application using DiaSim.

A. Demonstrated applications

We will demonstrate two applications to assist Henrick, a fictitious impaired person living autonomously in his apartment (Figure 4).

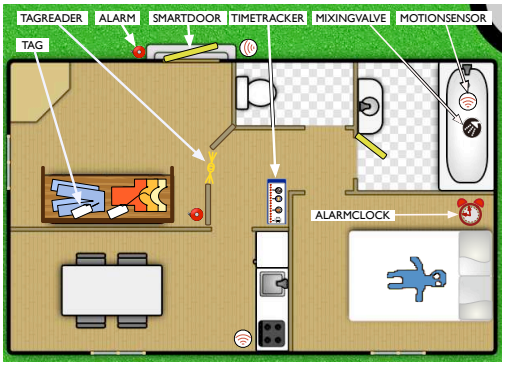

Figure 4. A 2D model of Henrick's apartment

The first application aims to assist Henrick to taking a shower. It involves the following entities: a motion detector, a smart door, a mixing valve, and a time tracker. If Henrick has been detected in the bathroom and the bathroom door is closed, then the mixingvalve regulates the water temperature to 35 degrees Celsius and then runs automatically. If Henrick is not under the shower, then the mixing valve is stopped. If the mixing valve is running, the time tracker automatically removes the shower task from its registered tasks.

The goal of the second application is to control the lights of Henrick's apartment. It relies on the following entities: lights, motion detectors, and an outside light sensor. If the luminosity is lower than a predefined threshold, the lights of a given room are turned on when Henrick enters this room, and turned off, when the room is empty.

B. Demonstration Steps

1) Creating a concrete environment: The first step of our demonstration shows how to create a concrete environment for which orchestration rules will be developed. It consists of (1) importing an existing Pantagruel taxonomy, (2) creating entities in a predefined 2D model of a house, described with structural characteristics (*i.e.,*walls and areas). This concrete environment is created using the DiaSim editor; it enables to locate a person moving in the different rooms of the house.

Figure 5. Edition of Henrick's apartment using DiaSim

2) Creating Pantagruel programs: In this step, we show how the Pantagruel orchestration editor is parameterized by the taxonomy, guiding the developer in the creation of programs that are syntactically correct and well-typed. As presented in Figure 6, each graphical element is associated with a contextual help, allowing the developer to write conditions and actions according to the available interface in the taxonomy. The Pantagruel visual editor is also connected to DiaSim, allowing entities created in the 2D model to be available in the editor.

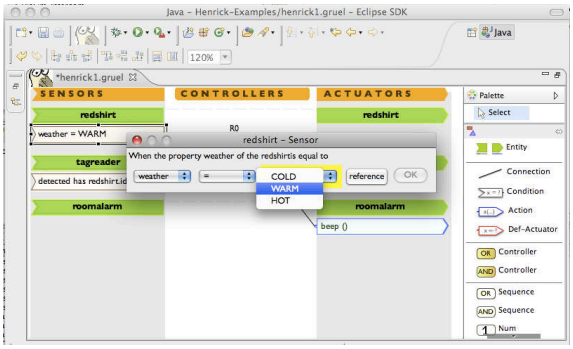

Figure 6. Orchestration guided by the taxonomy

3) Compiling and simulating Pantagruel programs: This step shows the compilation of a Pantagruel program presented in the previous step. The generated program is then given to DiaSim in order to be simulated. To do so, a predefined library of simulated and actual entities is provided, implementing the entity interaction interfaces, as declared in the Pantagruel taxonomy.

References

[1] W. Jouve *et al.*, "High-level programming support for robust pervasive computing applications," in *PerCom*, 2008.

[2] Z. Drey *et al.*, "A taxonomy-driven approach to visually prototyping pervasive computing applications," in *DSL-WC*, 2009.

[3] J. Bruneau *et al.*, "Diasim, a parameterized simulator for pervasive computing applications," in *Mobiquitous*, 2009.